



Queensland University of Technology
Brisbane Australia

This is the author's version of a work that was submitted/accepted for publication in the following source:

Rasmussen, Rune K. & Brown, Ross A. (2012) A deductive system for proving workflow models from operational procedures. *Future Generation Computer Systems*. (In Press)

This file was downloaded from: <http://eprints.qut.edu.au/48181/>

© Copyright 2012 Elsevier

Notice: *Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:*

<http://dx.doi.org/10.1016/j.future.2012.01.001>

A Deductive System for Proving Workflow Models from Operational Procedures

Rune Rasmussen, Ross Brown

*Mathematical, Information and Physical Sciences (MIPS)
Faculty of Science and Engineering,
Queensland University of Technology,
GPO Box 2434, Brisbane QLD 4000, Australia*

Abstract

Many modern business environments employ software to automate the delivery of workflows; whereas, workflow design and generation remains a laborious technical task for domain specialists. Several different approaches have been proposed for deriving workflow models. Some approaches rely on process data mining approaches, whereas others have proposed derivations of workflow models from operational structures, domain specific knowledge or workflow model compositions from knowledge-bases. Many approaches draw on principles from automatic planning, but conceptual in context and lack mathematical justification. In this paper we present a mathematical framework for deducing tasks in workflow models from plans in mechanistic or strongly controlled work environments, with a focus around automatic plan generations. In addition, we prove an associative composition operator that permits crisp hierarchical task compositions for workflow models through a set of mathematical deduction rules. The result is a logical framework that can be used to prove tasks in workflow hierarchies from operational information about work processes and machine configurations in controlled or mechanistic work environments.

Keywords: Workflow, Planning, Petri-net, Automation, Management, Modelling

1. Introduction

In structured business environments formalisms may be applied to inform workflow management. In modern environments, many organisations already manage workflow through computer programs known as *workflow management systems* (WfMS). WfMS emerged in the mid 90s when Van der Aalst et al. argued in [1] in favour of conceptual formalisms for workflow management systems, which lead to an initiative (see [2]) to catalogue workflow patterns towards formalising WfMS called the *Workflow Patterns Initiative*¹.

An outcome of the Pattern Initiative was YAWL (*Yet Another Workflow Language*), which is a formal language that has been developed around state-transition systems called Petri nets, to define control procedures in workflow management [3]. In the same context as other workflow languages such as the Business Process Execution Language (BPEL) [4] or scientific workflow languages such as Simple Conceptual Unified Flow Language (SCUFL) that is executable on the Taverna or myGrid software tools [5], YAWL is a model definition language that can be executed on the YAWL engine [6]. The YAWL engine, which has been developed from surveys of other reputable WfMS [2][7], involves an editor that permits users to create workflow model diagrams as annotated and directed networks of motifs, where each motif represents a control rule in the YAWL language. In delivery of workflow signals, the YAWL engine can convert workflow model diagrams to Petri-net based state machines that inform control signals and data exchanges through network

Email addresses: `r.rasmussen@qut.edu.au` (Rune Rasmussen), `r.brown@qut.edu.au` (Ross Brown)

¹Additional information about this initiative can be found at URL: www.workflowpatterns.com

interfaces with workflow events. In summary, this style of WfMS automates workflow through formally specified programs. The benefit of this style of WfMS is certainly workflow automation; however at the horizon of this automation, workflow language experts are still required for workflow model development. With that summary we arrive at the problem for this paper, which is how tasks in workflow models can be algebraically defined and applied to extend the horizon of automation for WfMS.

In controlled work environments, problems of workflow modelling may be sufficiently consistent to be solved with mathematical methods. An example can be seen in [8] with the problem of work allocations and peak load management for cloud servers, where the problems could be solved as constraint satisfaction problems. Even systems based on semantics and conceptual conventions such as web ontologies can have support for concrete knowledge systems improved with mathematical deduction and predicate logic [9], which could conceivably be applied to workflow systems. The need for mathematical systems to solve workflow modelling problems can be further seen where the analysis and deduction of workflows have been done in agent-based model simulations. In [10], very precise details of the actions and events have been mathematically defined for a set of workflow agents and logical systems of deduction and induction, such as ProM process mining tool (see [11]), were applied to recover workflow models for analysis. In this paper we present an approach using mathematical deduction to algebraically derive tasks for workflow models from resource data in work environments that may be characterised as mechanistic or involving sufficiently detailed tasks for a logical deductive approach. This approach can avoid some of the errors that probabilistic process mining methods can introduce to derived workflow models. Moreover, the deductive process we present here operate on templates of tasks, where those templates can be transported to and mathematically tested in different work environments, which permits efficient reuse of task templates.

Many authors have considered the problem of automatic workflow generation or process mining for workflow models. In addition to ProM and other process mining approaches in [11], van der Aalst in [12] proposed that *Bills-of-Materials* (BOMs) could be used to establish a template for workflow. In this context, a BOM is a hierarchically ordered set of materials that must be assembled in order to create a product, where the assembly steps characterise the work that must be done in a workflow. In this paper, we have developed a mathematical framework that can be applied to hierarchically deduce task templates for workflow models from operational level attributes. Our framework addresses several problems associated with current approaches. Process mining that relies on probabilistic approaches can involve errors that can be avoided in cases where workflow tasks can be derived algebraically. The transplanting of workflow models to different work environments can involve additional tailoring of the model, where the task templates defined in our systems can be efficiently tested for inclusion into different work environments and thus can be used to minimise workflow tailoring processes. In addition, our framework addresses the problem of deducing high-level task templates from plans, derived through automatic planning algorithms such as STRIPS, that can be tested in other work environments. In our framework, we have formalised a template construct about Petri nets that allows rapid testing and automatic recombinations and deductions of workflow tasks. In arriving at the particular details for our framework, Section 2 provides a summary of Petri nets and Petri-net based WfMS along with a multidimensional form of Petri nets called k -PN underpinning our framework; Section 3 visits automatic planning through discussion about STRIPS as a conduit for algebraic workflow deduction; Section 4 develops an isomorphism between logical operators used in automatic planning and additive operators defined for a class of k -PN; in addition, this section extends k -PNs to templates isomorphic to action operators used in STRIPS, provides and proves a serial combination operator for these templates and proves the associativity of this operator for hierarchical compositions; Section 5 looks at some practical applications and consequences for our framework; Section 6 gives a summary of the framework and its applications to discuss additional problems and further work.

2. Workflow Management Systems

YAWL is a very expressive workflow language based on rigorous analyses of existing workflow management systems and workflow languages [3][13]. The Workflow Pattern Initiative has evaluated several workflow products and have found considerable differences in their expressive powers. YAWL is a workflow

language that has evolved around a class of state-transition systems called *Petri nets* for their suitable levels of expressivity in representing workflows.

2.1. Petri Net Based Workflow Systems

Petri nets are state-transition systems that belong to a class of *nets* called *elementary nets* [14].

Definition 1. A net is a triple $N = (P, T, F)$ where:

1. P is a set of states, called *places*.
2. T is a set of state *transitions*.
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of flow relations called *arcs* between places and transitions (and between transitions and places).
4. For every $t \in T$ there exist $p, q \in P$ so that $(p, t), (t, q) \in F$ and for every $p, q \in P$, if $(p, t), (t, q) \in F$ then $p \neq q$.

The set $P \cup T$ are the net *elements*. Nets may be extended by colouring places to represent state, where such a colouring is known as a *configuration*.

Definition 2. An elementary net is a net of the form $EN = (N, M)$ where:

1. $N = (P, T, F)$ is a net.
2. $M : P \rightarrow \{0, 1, \dots\}$ is a configuration.

Definition 3. A Petri net is an elementary net of the form $PN = (EN, W)$, which has been extended with an arc labelling so that:

1. $EN = (P, T, F, M)$ is an elementary net.
2. $W : F \rightarrow \{1, 2, \dots\}$ is an arc labelling function that labels arcs with *multiplicity* values.

A common mode of communication for Petri nets is through diagrams. In a Petri net diagram, a place is conventionally described with a circle and may have a configuration of zero or more black dots called *tokens* (see left of Figure 1). In addition, a transition is conventionally described using a narrow rectangle (see right of Figure 1).

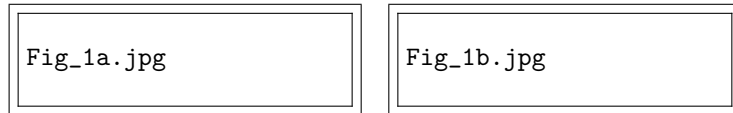


Figure 1: The left figure shows a place with a configuration of two tokens and single input and output arcs. The right figure shows a transition at the interface of arcs from input and output places.

A Petri net diagram conforms to a weighted and directed bipartite graph, where the elements $P \cup T$ can be partitioned so that P is one partition and T is the other partition. In addition, F is the set of arcs so that no arc directly connects two nodes from the same partition and each arc has a multiplicity that conforms to a mapping defined for W . In Figure 2, the place p_1 is an input place of transition t as there is an arc from p_1 to t ; whereas, the place p_2 is an output place of transition t as there is an arc from t to p_2 . Furthermore, let PN_0 denote a Petri net PN with an initial configuration M_0 and PN_1 denote the same Petri net PN with the configuration M_1 . In this example, PN_0 has its configuration M_0 given by the tokens on the left of Figure 2. This configuration *enables* transition t through the property that all input places have tokens “equal to or greater” than the multiplicities on their respective arcs to t . A transition *fires* by consuming tokens from its input places equal to the multiplicity of the respective input arcs and producing new tokens at the output places equal to the multiplicity of the respective output arcs; however, a transition can only fire if it is enabled. In Figure 2, the firing of transition t completes the configuration map from M_0 to M_1 .

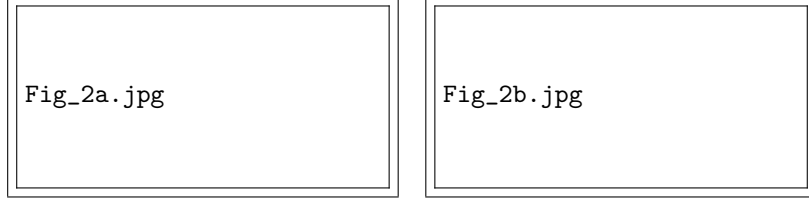


Figure 2: Since transition t is enabled, it fires and completes the configuration map from M_0 to M_1 .

Remark 1. Multiplicity mappings and markings in the previous definitions should not be interpreted as mappings to \mathbb{N} , the set of natural numbers. As the transition t fires in the previous example, tokens are subtracted from the input places and accumulated at the output places. The monoid $(\mathbb{N}, +)$ is well-defined, which implies that accumulations of tokens at the output places are also well-defined; however, subtraction is not closed-under operation for the natural numbers. For example, $2 - 5 = -3$ and $-3 \notin \mathbb{N}$, which implies that token subtraction is not well-defined. Even though Petri nets impose constraints that can avoid subtractive violations of closure, such constraints do not formally constitute a closure of \mathbb{N} under subtraction. This paper will adhere to a following more rigorous set of definitions for multiplicity mappings and markings:

1. Z is a closure of \mathbb{N} under addition (that may include modulo addition).
2. $W : F \rightarrow Z$ is a function that relates arcs to multiplicity values.
3. $M : P \rightarrow Z$ is a configuration.

In the context of workflow patterns, a *workflow model* can be defined using a Petri net, so that the transitions represent primitive tasks and the configurations represent workflow states [15]. Many workflow patterns, such as those found in the Workflow Pattern Initiative, involve workflow models based on Petri nets [2]. The models in Figure 3 are four of the most basic and widely used patterns for workflows. With the AND join (Figure 3 (a)), the target transition will only fire if both input places have received a token; however with the AND split (Figure 3 (b)), if the source transition fires then both places receive a token that enables both target transitions. With the OR join (Figure 3 (c)), either one or both source transitions may fire so that the place has tokens that enable the target transition; whereas with the FORK relation (Figure 3 (d)), either target transition is enabled and fires if and only if the place receives a token (this will depend on which transition can fire first).

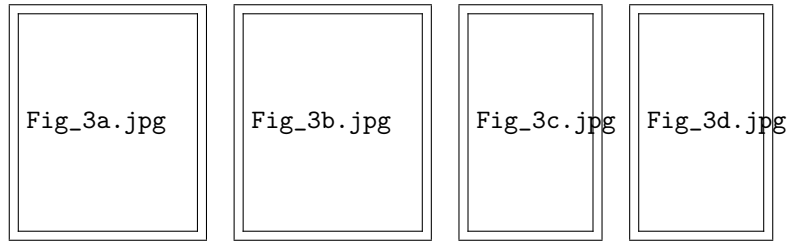


Figure 3: The four basic workflow patterns are: (a) AND join, (b) AND split, (c) OR join, (d) FORK relation.

The YAWL workflow language is a good example of a Petri-net based workflow modelling language. YAWL involves a set of symbols (see Figure 4), where some of the symbols directly represent the basic workflow patterns in Figure 3 and others represent some extensions of those patterns to facilitate different OR relationships.

The YAWL symbols divide into *task* and *condition* categories. A *task* (a box shape) is a logical unit of activity in a workflow and a *condition* (a circle shape) is a state that has conditional transitions. A YAWL workflow *specification* is a network of tasks and conditions, where exactly one *input condition* is at the start of a specification and exactly one *output condition* is at the end [16].

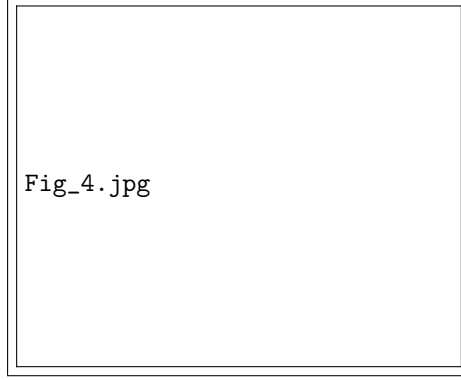


Figure 4: The original symbol set for YAWL.

2.2. High-Level Petri Nets

Jensen in [17] described a *Coloured* Petri net (CP-Net) as a multi-set variation on Petri nets. Jensen's definition of CP-Nets in [18] included guard functions on the transitions and multiplicity expressions on the arcs that can take net configurations as input. The CP-Net approach adds a level of flexibility that has permitted systems such as the “new” YAWL workflow language to express constructs based on CP-Nets for the management of concurrency, thread safety and reachability [3]. For the purpose of this paper however, we will define an extended variation of the classical Petri net involving the same principle of multidimensionality as CP-nets, but without features such as guard and multiplicity functions. We will show later in this paper that our extended Petri net can be deduced from plans derived from automatic planning algorithms.

Definition 4. A *k-dimensional Petri net* (k-PN) is a net of the form $PN^k = (N, W^k, M^k)$ where:

1. $N = (P, T, F)$ is a net.
2. $W^k : F \rightarrow Z_1 \times Z_2 \times \dots \times Z_k$ is a mapping from the arcs to a vector space of *k-dimensional multiplicity* values; where in each dimension i , Z_i is a closure of \mathbb{N} under addition.
3. $M^k : P \rightarrow Z_1 \times Z_2 \times \dots \times Z_k$ is a configuration of *k-dimensional* values.

For the purpose of k-PNs, a vector $\mathbf{a} \in Z_1 \times Z_2 \times \dots \times Z_k$ is “equal to or greater than” a vector $\mathbf{b} \in Z_1 \times Z_2 \times \dots \times Z_k$ if and only if $\bigwedge_{i=1}^k a_i \geq b_i$. Given that the only difference between the Petri net in Definition 3 and the k-dimensional Petri net defined here is dimensionality, a k-dimensional Petri net is a Petri net whenever $k = 1$.

In Figure 5, PN_0^3 denotes a 3-dimensional Petri net with an initial configuration M_0^3 and PN_1^3 denotes the same Petri net with the configuration M_1^3 . In this example, PN_0^3 has its configuration M_0^3 given by the 3-vectors shown in Figure 5 left. This configuration *enables* transition t through the property that all input places have vectors “equal to or greater than” the multiplicity vectors on their respective arcs to t according to the definition for “equal to or greater than” above. The transition *fires* by subtracting the multiplicity vectors at the input arcs from vectors at the respective input places and summing the multiplicity vectors at the output arcs to the vectors at the respective output places; however, a transition can only fire if it is enabled. In Figure 5, the firing of transition t completes the configuration map from M_0^3 to M_1^3 .

3. Automatic Planning

Workflow management systems that rely on workflow languages also rely on human input to encode workflow models. Human encoding implies a level of expertise that may be required many times in optimising a workflow; whereas, some kind of automation for this process could reduce or eliminate cost. Towards that end, this section looks at automatic planning as a candidate approach for algebraic workflow deduction.

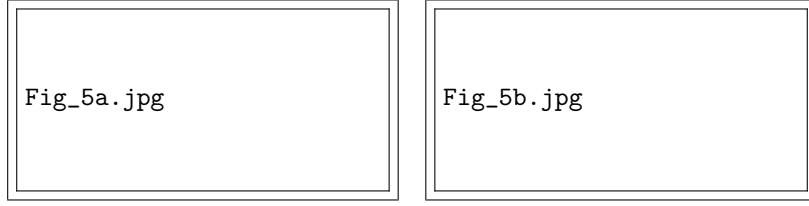


Figure 5: Since transition t is enabled, it fires and completes the configuration map from M_0^3 to M_1^3 .

Automatic planning deals with knowledge representations for actions and the problem of finding a sequence of actions or a *plan* that can transform a given world state to a desired goal state [19]. A good example of a planning problem that can be solved using automatic planners is the classical “Tower of Hanoi” problem, which was first proposed by Édouard Lucas in 1883. Our industrialised variation of the story for this problem is as follows:

Example 1. *In a certain large-scale electric motor rewind company, a tradesperson directed his apprentice to a rack consisting of three posts where only the first post was not empty, as it held three insulated copper coils of different sizes stacked largest to smallest. The tradesperson explained to the apprentice that the coils were stacked this way because the windings of a larger coil would slip apart should it ever be placed on top of a smaller coil, which would result in many hours of lost production as each coil is wound under pressure and cannot be corrected once the windings have slipped apart. The tradesperson then instructed the apprentice to move the coils from the first post to the last post according to the following rules:*

1. *Only one coil can be moved at a time*
2. *The coils can only be moved between the three posts*
3. *Smaller coils can sit on top of larger coils; however, a larger coil must never sit on top of a smaller coil.*

Operational level problems like this one generally require some planning; one may expect that the apprentice in the story will draw up a plan that resembles the solution given in Figure 6.

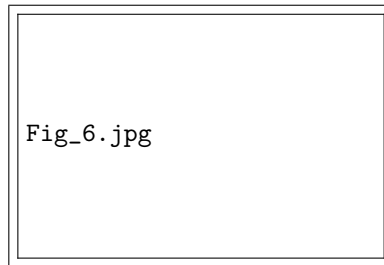


Figure 6: A plan that solves the “Tower of Hanoi” problem.

3.1. A Generalisation of the STanford Research Institute Problem Solver

The problem in the “Tower of Hanoi” example may even be solved on a computer; moreover, whole problem domains can be formally specified so that planning problems can be solved in general by computers. The *Stanford Research Institute Problem Solver* (STRIPS) is a planning language that was proposed in the early 1970s by Nilsson and Fikes in [20] to generalise automatic planning within given problem domains. The STRIPS language defines a class of planning operators that can be used to search world models for a plan. In this context, a world *model* is a well-formed formula in conjunctive normal form (CNF), where each

term is a positive ground predicate. Given a set of models M_Δ , STRIPS planners can search M_Δ by relating models through successor functions, where each successor function relates to a set of actions. In defining successor functions, the actions are represented by an operator, which we have called an *action operator*.

Definition 5. Given a set of models M_Δ , an *action operator* is a function of the form $op(p, e, \cdot) : M_\Delta \rightarrow M_\Delta$ where:

1. p is the action *precondition* and is a well-formed formula in CNF with terms that are positive ground predicates.
2. $e = (a, b)$ defines the action *effect* where a is a well-formed formula in CNF of positive ground predicates and b is a well-formed formula in CNF of negative ground predicates; so that, the expression $a \wedge b$ is the post condition of the action. That is, the set of predicates in a and the set of predicates in b are disjoint sets.

A set of predicates can always be defined so that models are in CNF and with terms that are positive ground predicates. For example, positive predicates for a switch z may be $\text{isOn}(z)$ or $\text{isOff}(z)$, even though there may be a bijection between $\neg \text{isOn}(z)$ and $\text{isOff}(z)$ due to the physical properties of the switch; $\text{isOn}(z)$ and $\text{isOff}(z)$ are both positive predicates. Dependencies between predicates in the context of physical states can be encoded in the effects of actions; for example, the effect of an action that activates a switch z could be $e = (a, b)$ where $a = \text{isOn}(z)$ and $b = \neg \text{isOff}(z)$. If needed, this arrangement allows for any physical dependencies imposed on a set of predicate symbols to be completely expressed in the effects of action operators, independently of any particular predicate symbolism.

Let $\phi(f)$ be the set of terms in a well-formed formula f , where f is either in disjunctive normal form (DNF) or in CNF. Where m is a model, the set $\phi(m)$ is called a *model set*. Given the complete set of models \mathcal{M} over a set of predicates, for any model $m \in \mathcal{M}$ the action operation $op(p, e, m)$ has the following formula:

$$op(p, e, m) = \begin{cases} \text{False} & \text{if } \phi(p) \not\subseteq \phi(m) \\ \bigwedge_{x \in ((\phi(m) - \phi(\neg b)) \cup \phi(a))} x & \text{otherwise} \end{cases} \quad (1)$$

In the case “otherwise” where $\phi(p) \subseteq \phi(m)$, Equation (1) can be interpreted as the model m minus the terms that are no longer true, plus the terms that are now true, given an action. For example, given precondition $p = A$ and effect $e = (B \wedge D, \neg A)$, $op(A, (B \wedge D, \neg A), B \wedge C) = \text{False}$ because $\{A\} \not\subseteq \{B, C\}$; that is, the precondition A has not been satisfied by the model $B \wedge C$. However, $op(A, (B \wedge D, \neg A), A \wedge C) = C \wedge B \wedge D$ because $\{A\} \subseteq \{A, C\}$ certifies that the precondition was satisfied by the model $A \wedge C$, $\phi(m) - \phi(\neg b) = \{A, C\} - \{A\} = \{C\}$ so therefore $((\phi(m) - \phi(\neg b)) \cup \phi(a)) = \{C\} \cup \{B, D\} = \{C, B, D\}$ and $\bigwedge_{x \in ((\phi(m) - \phi(\neg b)) \cup \phi(a))} x = C \wedge B \wedge D$.

3.2. A Rudimentary STRIPS Search Overview

A rudimentary form of STRIPS may be defined as a function $STRIPS = (O, m, g)$ that returns a list of action operations representing a plan, given O is a set of action operators, m is the initial world model and g is the goal model so that $m \neq g$. If the search begins from m , then its search space is a tree of models with m at the root (see Figure 7). For any initial model m , the search can generate the successors of m from the action operators, where the successors or *children* are defined: $\{op(p, e, m) \mid op(p, e, m) \in O, op(p, e, m) \neq \text{False}\}$. The leaf nodes are either the goal model g or those models where every action operation returns *False*. If the search finds g in a set of children then there exists a backtrack from g to m , where the list of operations along the track represents a plan. If no leaf node is the goal model then no plan exists and $STRIPS = (O, m, g)$ returns the empty plan. In conventional STRIPS planners, the STRIPS language ensures that parent-child relationships are also labelled with action commands for an agent’s actuators, so that a search can return a list of action commands representing a plan.

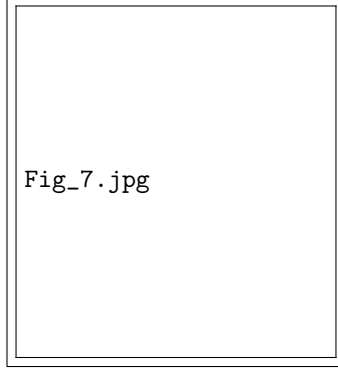


Figure 7: A STRIPS search may begin with the initial world model m at the root and search a model tree defined by actions operators until it find the goal model g , where it can trace a parent-wise path representative of a plan.

4. A k-PN Template for Action Operators

In a k predicate world, an action operation may be modelled using a k-PN that involves single input and output places configured with vectors that represent the input and output models of action operations; we will call such a k-PN an *action k-PN*. In this context, we will continue to use the definitions for model and model to set function ϕ from section 3.1. We have chosen an arrangement for action k-PN where the vector elements have a one-to-one correspondence with the k predicates and each element in a k -vector has the domain $\{0, \rho\}$ that represents either the absence (0) or presence (ρ) of a predicate in a model set. In addition, $\delta_k(s)$ is a function that maps any subset s of the k predicates to its k -vector representation. For example, if $k = 3$ and the predicates are A, B and C then $\delta_3(\{A, C\}) = \langle \rho, 0, \rho \rangle$.

4.1. Action k-PN

An action k-PN is a k-PN with arc to multiplicity mappings $W^k : F \rightarrow \mathbb{G}^k$ and configurations $M^k : P \rightarrow \mathbb{G}^k$, where \mathbb{G} is the algebraic field $\mathbb{G} = (\{0, \rho\}, +, \times)$ isomorphic to the \mathbb{Z}_2 field. The additive operator in \mathbb{G} is defined²:

$+$	0	ρ
0	0	ρ
ρ	ρ	0

(2)

Whereas, the multiplicative operator has been defined:

\times	0	ρ
0	0	0
ρ	0	ρ

(3)

4.1.1. Group Actions on Model Sets

The motive for choosing \mathbb{G} as the algebraic base for action k-PN has been to impose set operations on the model sets through group actions using vectors in \mathbb{G}^k . For a simple example, we might have model sets $\phi(m_x) = \{A, B\}$ and $\phi(m_y) = \{C\}$ in a world with the three predicates $\{A, B, C\}$, so that $\mathbf{m}_x = \delta_3(\phi(m_x)) = \langle \rho, \rho, 0 \rangle$ and $\mathbf{m}_y = \delta_3(\phi(m_y)) = \langle 0, 0, \rho \rangle$. Since $\phi(m_x)$ and $\phi(m_y)$ are disjoint sets, the vector sum $\mathbf{m}_x + \mathbf{m}_y$ can impose the set union $\phi(m_x) \cup \phi(m_y)$ through a group action on the sets, because

²Since \mathbb{G} satisfies that same Dedekind-Peano axioms as the set of natural numbers under base-2 modulo addition and multiplication, \mathbb{G} is a closure of \mathbb{N} under both operations.

$\mathbf{m}_x + \mathbf{m}_y = \langle \rho, \rho, 0 \rangle + \langle 0, 0, \rho \rangle = \langle \rho, \rho, \rho \rangle$ and $\delta_k^{-1}(\langle \rho, \rho, \rho \rangle) = \{A, B, C\}$, where δ_3^{-1} is the inverse operation of δ_3 that completes the group action equivalence $\{A, B\} \cup \{C\} = \{A, B, C\}$.

In addition, the set subtraction $\phi(m_x) - \phi(m_y)$ involves the group action:

$$\mathbf{m}_x + \mathbf{m}_x * \mathbf{m}_y, \text{ where } \mathbf{m}_x * \mathbf{m}_y \text{ is defined } \langle \mathbf{m}_x[i] \times \mathbf{m}_y[i] \rangle_{0 \leq i \leq k} \quad (4)$$

For example, given model sets $\phi(m_x) = \{A, B\}$ and $\phi(m_y) = \{A, C\}$ in a three predicate world $\{A, B, C\}$:

$$\begin{aligned} \mathbf{m}_x + \mathbf{m}_x * \mathbf{m}_y &= \delta_3(\phi(m_x)) + \delta_3(\phi(m_x)) * \delta_3(\phi(m_y)) \\ &= \langle \rho, \rho, 0 \rangle + (\langle \rho, \rho, 0 \rangle * \langle \rho, 0, \rho \rangle) \\ &= \langle \rho, \rho, 0 \rangle + (\langle \rho, 0, 0 \rangle) \\ &= \langle 0, \rho, 0 \rangle \\ \delta_3^{-1}(\langle 0, \rho, 0 \rangle) &= \{B\}, \text{ the realisation of } \{A, B\} - \{A, C\} \end{aligned}$$

Given a group action for set subtraction, set union $\phi(m_x) \cup \phi(m_y)$ in general involves the following group action:

$$\mathbf{m}_y + \mathbf{m}_x + \mathbf{m}_x * \mathbf{m}_y \quad (5)$$

For example given model sets $\phi(m_x) = \{A, B\}$ and $\phi(m_y) = \{A, C\}$ in a three predicate world $\{A, B, C\}$:

$$\begin{aligned} \mathbf{m}_y + \mathbf{m}_x + \mathbf{m}_x * \mathbf{m}_y &= \delta_3(\phi(m_y)) + \delta_3(\phi(m_x)) + \delta_3(\phi(m_x)) * \delta_3(\phi(m_y)) \\ &= \langle \rho, \rho, 0 \rangle + \langle \rho, 0, \rho \rangle + (\langle \rho, \rho, 0 \rangle * \langle \rho, 0, \rho \rangle) \\ &= \langle 0, \rho, \rho \rangle + (\langle \rho, 0, 0 \rangle) \\ &= \langle \rho, \rho, \rho \rangle \\ \delta_3^{-1}(\langle \rho, \rho, \rho \rangle) &= \{A, B, C\}, \text{ the realisation of } \{A, B\} \cup \{A, C\} \end{aligned}$$

It is easy to see that the expression $\mathbf{m}_x * \mathbf{m}_y$ is a group action for the intersection of sets $\phi(m_x) \cap \phi(m_y)$. Given the vector $\mathbf{m}_x = \delta_k(\phi(m_x)) = \langle \rho, \rho, 0 \rangle$, we may consider the complement vector $\neg \mathbf{m}_x = \delta_k(\phi(\neg m_x)) = \langle \rho, \rho, 0 \rangle + \langle \rho, \rho, \rho \rangle = \langle 0, 0, \rho \rangle$. In Section 3, b was defined as a well-formed formula in CNF of negative ground predicates, therefore $\neg b$ is a well-formed formula in DNF of positive ground predicates and $\neg \mathbf{b} = \delta_k(\phi(\neg b))$ is a vector where the predicates in $\neg b$ are (ρ) present. Table 1 gives a list of group actions and related set operations.

Table 1: Group actions using \mathbb{G}^k vectors to impose set operations on model sets \mathcal{A} , \mathcal{B} , and the universal model set \mathcal{U} of size k , where $\phi(m_A) = \mathcal{A}$, $\phi(m_B) = \mathcal{B}$ and $\phi(m_U) = \mathcal{U}$.

Group action rules	Delta expansions	Set Operations
$\neg \mathbf{m}_A, \mathbf{m}_A + \mathbf{m}_U$	$\delta_k(\phi(m_A)) + \delta_k(\phi(m_U))$	\mathcal{A}^c
$\mathbf{m}_A * \mathbf{m}_B$	$\delta_k(\phi(m_A)) * \delta_k(\phi(m_B))$	$\mathcal{A} \cap \mathcal{B}$
$\mathbf{m}_A + \mathbf{m}_A * \mathbf{m}_B$	$\delta_k(\phi(m_A)) + \delta_k(\phi(m_A)) * \delta_k(\phi(m_B))$	$\mathcal{A} - \mathcal{B}$
$\mathbf{m}_B + \mathbf{m}_A + \mathbf{m}_A * \mathbf{m}_B$	$\delta_k(\phi(m_B)) + \delta_k(\phi(m_A)) + \delta_k(\phi(m_A)) * \delta_k(\phi(m_B))$	$\mathcal{A} \cup \mathcal{B}$

4.2. A k -PN Template

Consider an action operation $op(p, e, m)$ where m satisfies the precondition $\phi(p) \subseteq \phi(m)$, the action operation transforms m to a model represented by the set $(\phi(m) - \phi(\neg b)) \cup \phi(a)$ according to Equation (1). This expression can be modelled by an action k -PN where the input place has a fork to two transitions, followed by their join at the output place (see Figure 8). One transition in the fork can generate a vector $\delta_k(\phi(a))$ that will represent predicates made positive by the action operation, whereas the other transition can generate a vector $\delta_k(\phi(m) - (\phi(\neg b) \cup \phi(a)))$ that will represent predicates unchanged by the action

operation. We will call these two transitions the *Action Transport* and the *Model Transport*, respectively. Since the sets $\phi(m) - (\phi(\neg b) \cup \phi(a))$ and $\phi(a)$ are disjoint, the vector sum $\delta_k(\phi(m) - (\phi(\neg b) \cup \phi(a))) + \delta_k(\phi(a))$ represents their union; moreover, since the set union $(\phi(m) - (\phi(\neg b) \cup \phi(a))) \cup \phi(a)$ can be simplified to the expression $(\phi(m) - \phi(\neg b)) \cup \phi(a)$ this vector sum represents the target set in Equation (1), which is the desired result in devising a model for action operators. The action and model transports in Figure 8 define separable layers that will simplify proofs for operators on k-PN templates later in this section; however, a k-PN template can be more compactly represented by a single transition between p_1 and p_2 with \mathbf{p} at the input and $\mathbf{a} + \mathbf{m}_\phi$ at the output. This separation may be of further use in that the model transport involves \mathbf{m}_ϕ , which represents predicates unchanged by the action operation and may support arguments for some additional action k-PN templates in parallel.

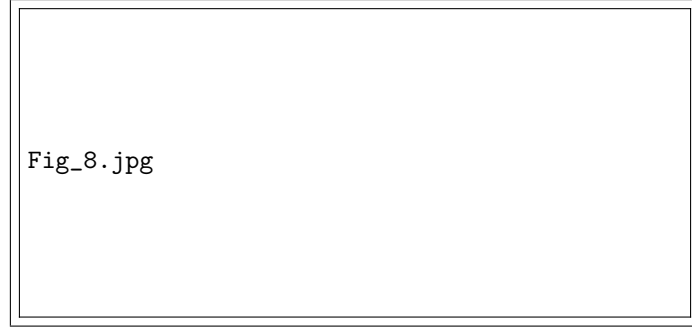


Figure 8: Action k-PN templates are operators based on k-PN that model action operators according to Definition 6.

An action k-PN can be generalised to an action k-PN *template* that models the functionality of an action operator as follows:

Definition 6. An *action k-PN template* $f_{kPN}(\mathbf{m}_x; \mathbf{p}, \mathbf{a}, \neg \mathbf{b}; \mathbf{m}_y)$ that models an action operator $m_y = op(p, e, m_x)$ with variables m_x and m_y , is a template for a k-PN of the form shown in Figure 8, with the following properties:

1. \mathbf{m}_x is a variable where $\mathbf{m}_x = \delta_k(\phi(m_x))$ for the variable m_x .
2. \mathbf{m}_y is a variable where $\mathbf{m}_y = \delta_k(\phi(m_y))$ for the variable m_y .
3. \mathbf{p} where $\mathbf{p} = \delta_k(\phi(p))$, is a vector value that represents the precondition p .
4. \mathbf{a} where $\mathbf{a} = \delta_k(\phi(a))$, is a vector value that represents the predicates that evaluate to *True* after an action operation, given to the effect $e = (a, b)$.
5. $\neg \mathbf{b}$ where $\neg \mathbf{b} = \delta_k(\phi(\neg b))$ is a vector value that represents the predicates that evaluate to *False* after an action operation, given to the effect $e = (a, b)$.
6. \mathbf{m}_ϕ is a variable where $\mathbf{m}_\phi = \delta_k(\phi(m_x) - (\phi(\neg b) \cup \phi(a)))$ and depends on \mathbf{m}_x through the formula:

$$\mathbf{m}_\phi = \mathbf{m}_x + \mathbf{m}_x * (\neg \mathbf{b} + \mathbf{a} + (\neg \mathbf{b} * \mathbf{a})) \quad (6)$$

In a k-PN template, a value assignment to \mathbf{m}_x allows the evaluation of \mathbf{m}_ϕ through Equation (6); the evaluations of both \mathbf{m}_x and \mathbf{m}_ϕ provides a *reduction* of the action k-PN template to an action k-PN. Both transition are enabled if \mathbf{m}_x takes a value that is “equal to or greater than” \mathbf{p} according to Definition 4 for action k-PN. If both transitions fire, then the net will assign the sum $\mathbf{m}_\phi + \mathbf{a}$ to the variable \mathbf{m}_y at place p_2 . Since the sets $\phi(m) - (\phi(\neg b) \cup \phi(a))$ and $\phi(a)$ are disjoint, their set union is sufficiently represented by the assignment $\mathbf{m}_y \leftarrow \mathbf{m}_\phi + \mathbf{a}$ at place p_2 . We will use the following predicate to test reductions of action k-PN templates:

Definition 7. The predicate $f_{kPN} \rightarrow (\mathbf{m}_{out} \mid \mathbf{m}_{in})$, which is *True* if and only if f_{kPN} reduces to an action k-PN that will fire and generate the value \mathbf{m}_{out} at the output place, given \mathbf{m}_{in} is the input value.

Since plans are sequences of actions, k-PN representations of plans will require operators that combine action k-PN templates. In this paper, we will consider operations that can be used to solve serial (\oplus) configuration problems for action k-PN templates, where two action k-PN templates in a *serial* configuration share a common place at the output of one template and at the input of the other. The serial (\oplus) operator returns a single action k-PN template that is equivalent to the serial combination of two k-PN templates.

For clarity in the following theorems, we have let the relationship between any k-vector and the model it represents be implied through a common symbol; for example, the k-vector \mathbf{x} will be understood to represent the model x through the common use of the symbol ‘ \mathbf{x} ’.

Theorem 1. *Given action k-PN templates $f_{kPN}(\mathbf{m}_x; \mathbf{p}_f, \mathbf{a}_f, \neg \mathbf{b}_f; \mathbf{m}_y)$, $g_{kPN}(\mathbf{m}_x; \mathbf{p}_g, \mathbf{a}_g, \neg \mathbf{b}_g; \mathbf{m})$ and $h_{kPN}(\mathbf{m}; \mathbf{p}_h, \mathbf{a}_h, \neg \mathbf{b}_h; \mathbf{m}_y)$; if and only if $\phi(\neg b_g) \cap \phi(p_h) = \emptyset$ then $f_{kPN} = g_{kPN} \oplus h_{kPN}$; reads, “ f_{kPN} is the serial combination of g_{kPN} and h_{kPN} ” and is defined:*

$$\phi(p_f) = \phi(p_g) \cup (\phi(p_h) - \phi(a_g)) \quad (7)$$

$$\phi(a_f) = \phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h)) \quad (8)$$

$$\phi(\neg b_f) = \phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h)) \quad (9)$$

PROOF OF THEOREM. Serial Combination of k-PN (\oplus):

1. **The precondition**, “if and only if $\phi(\neg b_g) \cap \phi(p_h) = \emptyset$ ”:
 - (a) Since $\phi(m) = \phi(a_g) \cup (\phi(m_x) - \phi(\neg b_g))$ and $\phi(a_g) \cap \phi(\neg b_g) = \emptyset$, it follows that $\phi(\neg b_g) \cap \phi(m) = \emptyset$.
 - (b) Since $\phi(\neg b_g) \cap \phi(m) = \emptyset$, if $\phi(\neg b_g) \cap \phi(p_h) \neq \emptyset$ then $\phi(p_h) \not\subset \phi(m)$. That is, the precondition of h_{kPN} will never be satisfied.
 - (c) since $\phi(\neg b_g) \cap \phi(m) = \emptyset$, if and only if $\phi(\neg b_g) \cap \phi(p_h) = \emptyset$ then $\phi(p_h) \subset \phi(m)$; the precondition of h_{kPN} can be satisfied.
2. **Proposition**, “ $\phi(p_f) = \phi(p_g) \cup (\phi(p_h) - \phi(a_g))$ ”:
 - (a) If the precondition $\phi(p_g) \subset \phi(m_x)$, then the precondition $\phi(p_f) \subset \phi(m_x)$ follows; since, \mathbf{m}_x is the vector at the inputs of both g_{kPN} and f_{kPN} .
 - (b) Assume $\phi(p_f) = \phi(p_g)$; if $\phi(p_f) = \phi(p_g)$, then \mathbf{m}_x may be such that there exist an \mathbf{m} so that $g_{kPN} \rightarrow (\mathbf{m} \mid \mathbf{m}_x)$, but $\phi(p_h) \not\subset \phi(m)$. That is, the precondition of the action for h_{kPN} is not satisfied.
 - (c) Assume $\phi(p_f) = \phi(p_g) \cup \phi(p_h)$; if $\phi(p_f) = \phi(p_g) \cup \phi(p_h)$ then \mathbf{m}_x may be such that there exist an \mathbf{m} so that $g_{kPN} \rightarrow (\mathbf{m} \mid \mathbf{m}_x)$ and $\phi(p_h) \subset \phi(m)$ given the precondition $\phi(\neg b_g) \cap \phi(p_h) = \emptyset$; however, g_{kPN} may generate \mathbf{a}_g so that $\phi(p_h) \subset C \cup \phi(a_g)$, where $C \subset \phi(m_x)$ and $C \neq \emptyset$.
 - (d) if $\phi(p_f) = \phi(p_g) \cup (\phi(p_h) - \phi(a_g))$ then \mathbf{m}_x may be such that there exist an \mathbf{m} so that $g_{kPN} \rightarrow (\mathbf{m} \mid \mathbf{m}_x)$ and $\phi(p_h) \subset \phi(m)$ given the precondition $\phi(\neg b_g) \cap \phi(p_h) = \emptyset$; however, g_{kPN} can only generate \mathbf{a}_g so that $\phi(p_h) \subset C \cup \phi(a_g)$ and $C = \emptyset$.
 - (e) All possible exceptions have been exhausted: $\phi(p_f) = \phi(p_g) \cup (\phi(p_h) - \phi(a_g))$.

3. **Propositions**, “ $\phi(a_f) = \phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h))$ ” and “ $\phi(\neg b_f) = \phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h))$ ”:

$$\begin{aligned}
\phi(m) &= (\phi(m_x) - \phi(\neg b_g) \cup \phi(a_g)) \cup \phi(a_g) \\
\phi(m_y) &= (\phi(m) - (\phi(\neg b_h) \cup \phi(a_h))) \cup \phi(a_h) \\
\phi(m_y) &= ((\phi(m_x) - (\phi(\neg b_g) \cup \phi(a_g))) \cup \phi(a_g) - (\phi(\neg b_h) \cup \phi(a_h))) \cup \phi(a_h) \\
\phi(m_y) &= (A \cup B - C) \cup \phi(a_h) \\
\phi(m_y) &= (A \cup (B - C) \cup (B \cap C) - C) \cup \phi(a_h) \\
\phi(m_y) &= (A \cup (B \cap C) - C \cup (B - C)) \cup (B - C) \cup \phi(a_h) \\
\phi(m_y) &= (A \cup (B \cap C) - C \cup B) \cup (B - C) \cup \phi(a_h) \\
\phi(m_y) &= (A - C \cup B) \cup (B - C) \cup \phi(a_h) \\
\phi(m_y) &= ((\phi(m_x) - (\phi(\neg b_g) \cup \phi(a_g))) - (\phi(\neg b_h) \cup \phi(a_h)) \cup \phi(a_g)) \cup \\
&\quad (\phi(a_g) - (\phi(\neg b_h) \cup \phi(a_h))) \cup \phi(a_h) \\
\phi(m_y) &= (\phi(m_x) - (\phi(\neg b_g) \cup \phi(\neg b_h) \cup \phi(a_g) \cup \phi(a_h))) \cup (\phi(a_g) - \phi(\neg b_h)) \cup \phi(a_h)
\end{aligned}$$

If $\phi(a_f) = (\phi(a_g) - \phi(\neg b_h)) \cup \phi(a_h)$ and $\phi(\neg b_f) = \phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h))$ then

$$\phi(\neg b_f) \cap \phi(a_f) = \emptyset$$

and

$$\phi(\neg b_f) \cup \phi(a_f) = \phi(\neg b_g) \cup \phi(\neg b_h) \cup \phi(a_g) \cup \phi(a_h)$$

therefore

$$\phi(m_y) = (\phi(m_x) - (\phi(\neg b_f) \cup \phi(a_f))) \cup \phi(a_f)$$

□

Theorem 2. The serial combination operator \oplus for action k -PN templates is associative.

PROOF OF THEOREM. **Proposition:** $(f_{kPN} \oplus g_{kPN}) \oplus h_{kPN} = f_{kPN} \oplus (g_{kPN} \oplus h_{kPN})$

Left Side: Let $\alpha_{kPN} = f_{kPN} \oplus g_{kPN}$ and $\beta_{kPN} = \alpha_{kPN} \oplus h_{kPN}$. The solution for α_{kPN} is:

$$\begin{aligned}
\phi(p_\alpha) &= \phi(p_f) \cup (\phi(p_g) - \phi(a_f)) \\
\phi(a_\alpha) &= \phi(a_g) \cup (\phi(a_f) - \phi(\neg b_g)) \\
\phi(\neg b_\alpha) &= \phi(\neg b_g) \cup (\phi(\neg b_f) - \phi(a_g))
\end{aligned}$$

And, the solution for β_{kPN} after substituting in α_{kPN} :

$$\phi(p_\beta) = \phi(p_f) \cup (\phi(p_g) - \phi(a_f)) \cup (\phi(p_h) - (\phi(a_g) \cup (\phi(a_f) - \phi(\neg b_g)))) \quad (10)$$

$$\phi(a_\beta) = \phi(a_h) \cup ((\phi(a_g) \cup (\phi(a_f) - \phi(\neg b_g))) - \phi(\neg b_h)) \quad (11)$$

$$\phi(\neg b_\beta) = \phi(\neg b_h) \cup ((\phi(\neg b_g) \cup (\phi(\neg b_f) - \phi(a_g))) - \phi(a_h)) \quad (12)$$

Right Side: Let $\gamma_{kPN} = g_{kPN} \oplus h_{kPN}$ and $\delta_{kPN} = f_{kPN} \oplus \gamma_{kPN}$. The solution for γ_{kPN} is:

$$\begin{aligned}
\phi(p_\gamma) &= \phi(p_g) \cup (\phi(p_h) - \phi(a_g)) \\
\phi(a_\gamma) &= \phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h)) \\
\phi(\neg b_\gamma) &= \phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h))
\end{aligned}$$

And, the solution for δ_{kPN} after substituting in γ_{kPN} :

$$\phi(p_\delta) = \phi(p_f) \cup ((\phi(p_g) \cup (\phi(p_h) - \phi(a_g))) - \phi(a_f)) \quad (13)$$

$$\phi(a_\delta) = \phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h)) \cup (\phi(a_f) - (\phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h)))) \quad (14)$$

$$\phi(\neg b_\delta) = \phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h)) \cup (\phi(\neg b_f) - (\phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h)))) \quad (15)$$

Left Side \equiv Right Side:

Equations (10) is equivalent to (13) :

$$\phi(p_\beta) = \phi(p_f) \cup (\phi(p_g) - \phi(a_f)) \cup (\phi(p_h) - (\phi(a_g) \cup (\phi(a_f) - \phi(\neg b_g))))$$

Since $\phi(\neg b_g) \cap \phi(p_h) = \emptyset$ is the precondition for serial combination $g_{kPN} \oplus h_{kPN}$, then:

$$\begin{aligned}\phi(p_\beta) &= \phi(p_f) \cup (\phi(p_g) - \phi(a_f)) \cup (\phi(p_h) - (\phi(a_g) \cup \phi(a_f))) \\ \phi(p_\beta) &= \phi(p_f) \cup (\phi(p_g) - \phi(a_f)) \cup ((\phi(p_h) - \phi(a_g)) - \phi(a_f)) \\ \phi(p_\beta) &= \phi(p_f) \cup ((\phi(p_g) \cup ((\phi(p_h) - \phi(a_g))) - \phi(a_f)) \\ \phi(p_\beta) &= \phi(p_\delta)\end{aligned}$$

Equations (11) is equivalent to (14) :

$$\begin{aligned}\phi(a_\delta) &= \phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h)) \cup (\phi(a_f) - (\phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h)))) \\ \phi(a_\delta) &= \phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h)) \cup (\phi(a_f) - (\phi(\neg b_h) \cup \phi(\neg b_g))) \\ \phi(a_\delta) &= \phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h)) \cup ((\phi(a_f) - \phi(\neg b_g)) - \phi(\neg b_h)) \\ \phi(a_\delta) &= \phi(a_h) \cup ((\phi(a_g) \cup (\phi(a_f) - \phi(\neg b_g))) - \phi(\neg b_h)) \\ \phi(a_\delta) &= \phi(a_\beta)\end{aligned}$$

Equations (12) is equivalent to (15) :

$$\begin{aligned}\phi(\neg b_\delta) &= \phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h)) \cup (\phi(\neg b_f) - (\phi(a_h) \cup (\phi(a_g) - \phi(\neg b_h)))) \\ \phi(\neg b_\delta) &= \phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h)) \cup (\phi(\neg b_f) - (\phi(a_h) \cup \phi(a_g))) \\ \phi(\neg b_\delta) &= \phi(\neg b_h) \cup (\phi(\neg b_g) - \phi(a_h)) \cup ((\phi(\neg b_f) - \phi(a_g)) - \phi(a_h)) \\ \phi(\neg b_\delta) &= \phi(\neg b_h) \cup ((\phi(\neg b_g) \cup (\phi(\neg b_f) - \phi(a_g))) - \phi(a_h)) \\ \phi(\neg b_\delta) &= \phi(\neg b_\beta)\end{aligned}$$

□

An Example of the Serial Combination Operator \oplus . In this example, let us assume that a world has been defined with a set of predicates $\{A, B, C, D, E, F, G\}$ and the initial world model $m_0 = A \wedge C \wedge E \wedge G$ maps to a vector $\mathbf{m}_0 = \delta_7(\phi(m_0)) = \langle \rho, 0, \rho, 0, \rho, 0, \rho \rangle$. In addition, consider action k-PN template $f_{kPN}(\mathbf{m}_x; \mathbf{p}_f, \mathbf{a}_f, \neg \mathbf{b}_f; \mathbf{m}_y)$ where:

1. $\mathbf{p}_f = \langle \rho, 0, \rho, 0, 0, 0, 0 \rangle$
2. $\mathbf{a}_f = \langle 0, 0, 0, \rho, \rho, 0, 0 \rangle$
3. $\neg \mathbf{b}_f = \langle \rho, 0, 0, 0, 0, 0, 0 \rangle$

and action k-PN template $g_{kPN}(\mathbf{m}_x; \mathbf{p}_g, \mathbf{a}_g, \neg \mathbf{b}_g; \mathbf{m}_y)$ where:

1. $\mathbf{p}_g = \langle 0, 0, 0, \rho, \rho, 0, \rho \rangle$
2. $\mathbf{a}_g = \langle 0, 0, 0, 0, 0, \rho, 0 \rangle$
3. $\neg \mathbf{b}_g = \langle 0, 0, 0, \rho, \rho, 0, 0 \rangle$

In applying f_{kPN} followed by g_{kPN} , the initial model vector \mathbf{m}_0 satisfies the precondition of f_{kPN} so $f_{kPN}(\mathbf{m}_0; \mathbf{p}_f, \mathbf{a}_f, \neg \mathbf{b}_f; \mathbf{m}_1)$ can be evaluated:

1. $\mathbf{a}_f = \langle 0, 0, 0, \rho, \rho, 0, 0 \rangle$
2. $\mathbf{m}_{f\phi} = \mathbf{m}_0 + \mathbf{m}_0 * (\neg \mathbf{b}_f + \mathbf{a}_f + (\neg \mathbf{b}_f * \mathbf{a}_f)) = \langle 0, 0, \rho, 0, 0, 0, \rho \rangle$
3. $\mathbf{m}_1 = \mathbf{a}_f + \mathbf{m}_{f\phi} = \langle 0, 0, \rho, \rho, \rho, 0, \rho \rangle$

Since \mathbf{m}_1 satisfies the precondition for g_{kPN} , $g_{kPN}(\mathbf{m}_1; \mathbf{p}_g, \mathbf{a}_g, \neg\mathbf{b}_g; \mathbf{m}_2)$ can be evaluated:

1. $\mathbf{a}_g = \langle 0, 0, 0, 0, 0, \rho, 0 \rangle$
2. $\mathbf{m}_{g\phi} = \mathbf{m}_1 + \mathbf{m}_1 * (\neg\mathbf{b}_g + \mathbf{a}_g + (\neg\mathbf{b}_g * \mathbf{a}_g)) = \langle 0, 0, \rho, 0, 0, 0, \rho \rangle$
3. $\mathbf{m}_2 = \mathbf{a}_g + \mathbf{m}_{g\phi} = \langle 0, 0, \rho, 0, 0, \rho, \rho \rangle$

Since $\neg\mathbf{b}_f + \mathbf{p}_g = \langle 0, 0, 0, 0, 0, 0, 0 \rangle$ means $\phi(\neg b_f) \cap \phi(p_g) = \emptyset$, the combination $h_{kPN} = f_{kPN} \oplus g_{kPN}$ can be evaluated using Theorem 1 as:

1. $\mathbf{p}_h = \phi(p_f) \cup (\phi(p_g) - \phi(a_f)) = \langle \rho, 0, \rho, 0, 0, 0, \rho \rangle$
2. $\mathbf{a}_h = \phi(a_g) \cup (\phi(a_f) - \phi(\neg b_g)) = \langle 0, 0, 0, 0, 0, \rho, 0 \rangle$
3. $\neg\mathbf{b}_h = \phi(\neg b_g) \cup (\phi(\neg b_f) - \phi(a_g)) = \langle \rho, 0, 0, \rho, \rho, 0, 0 \rangle$

As expected, the initial model vector \mathbf{m}_0 satisfies the precondition of h_{kPN} so $h_{kPN}(\mathbf{m}_0; \mathbf{p}_h, \mathbf{a}_h, \neg\mathbf{b}_h; \mathbf{m}_3)$ can be evaluated:

1. $\mathbf{a}_h = \langle 0, 0, 0, 0, 0, \rho, 0 \rangle$
2. $\mathbf{m}_{h\phi} = \mathbf{m}_0 + \mathbf{m}_0 * (\neg\mathbf{b}_h + \mathbf{a}_h + (\neg\mathbf{b}_h * \mathbf{a}_h)) = \langle 0, 0, \rho, 0, 0, 0, \rho \rangle$
3. $\mathbf{m}_3 = \mathbf{a}_h + \mathbf{m}_{h\phi} = \langle 0, 0, \rho, 0, 0, \rho, \rho \rangle$

Finally, $\mathbf{m}_2 = \mathbf{m}_3$, which shows that the combination $h_{kPN} = f_{kPN} \oplus g_{kPN}$ permits an evaluation from \mathbf{m}_0 to \mathbf{m}_2 without the cost of evaluating \mathbf{m}_1 .

4.3. A Plan to k-PN Mapping Procedure

The search that was presented in Section 3.2 could be configured to return a list of models in place of a list of action operations; let \overline{STRIPS} be such a search algorithm. Given a k predicate world and a list of models $M_\Sigma = [m_0, m_1, m_2, \dots, m_i, \dots, m_n]$ from a \overline{STRIPS} search, where m_0 is the root model and m_n is the goal model. A mapping from the list M_Σ to a list of action k-PN templates can be achieved with the following steps:

1. Generate a set \mathcal{A} of action k-PN templates (conforming to Definition 6) from the action operators that were available to the \overline{STRIPS} search.
2. Define a list MV of k -vectors, so that $MV = [\delta_k(\phi(m_i)) \mid m_i \in M_\Sigma]$.
3. Create a list A_{MV} of action k-PN templates for pairwise k -vectors in MV :
 - (a) Initialise A_{MV} to an empty list.
 - (b) For $i = 0$ to n
 - (c) Get a list $[\mathbf{m}_i, \mathbf{m}_{i+1}]$ of vectors, where $[\mathbf{m}_i, \mathbf{m}_{i+1}] \subset MV$.
 - (d) Find the exact action k-PN template f_{kPN} , where $f_{kPN} \in \mathcal{A}$ and $f_{kPN} \rightarrow (\mathbf{m}_{i+1} \mid \mathbf{m}_i)$.
 - (e) Append the list A_{MV} so that $A_{MV} \leftarrow A_{MV} \cup [f_{kPN}]$.
 - (f) End For
 - (g) Return the list A_{MV} .

On termination of this algorithm, the list A_{MV} will contain the necessary action k-PN templates required to represent the plan associated with the models in M_Σ . From here forward and into the next section, plans will be described in terms of action k-PN template lists.

4.4. Discussion

With respect to the plan to k-PN Mapping Procedure, unlike a plan returned in a STRIPS search, the templates in the list A_{MV} are independent of world state; that is, the action k-PN templates in this list may reduce to k-PN for many possible input values and can therefore serve as a plan template. In addition, the reduction of action k-PN templates in a list A_{MV} given the correct inputs will return actions k-PNs that can be used to form a Petri Net model of a plan, which is essentially a workflow model. Finally, the associative property of the \oplus operator implies that serial combinations over several action k-PN templates can be extended hierarchically. For example, assume we have a list of action k-PN templates: $A_{MV} =$

$[f_{kPN,0}, f_{kPN,1}, f_{kPN,2}, f_{kPN,3}, f_{kPN,4}, f_{kPN,5}]$ that represents a plan. The single action k-PN template F_{kPN} that also represents the plan can be computed:

$$F_{kPN} = f_{kPN,0} \oplus f_{kPN,1} \oplus f_{kPN,2} \oplus f_{kPN,3} \oplus f_{kPN,4} \oplus f_{kPN,5}$$

Or given the associativity of \oplus , it may be computed with any possible precedence nesting. For example,

$$F_{kPN} = (((f_{kPN,0} \oplus f_{kPN,1}) \oplus (f_{kPN,2} \oplus f_{kPN,3})) \oplus (f_{kPN,4} \oplus f_{kPN,5}))$$

Which may be represented by the following hierarchy:



Figure 9: A hierarchical composition of a serial combination of six action k-PN templates through a system of associations.

5. A System of Workflow Model Deductions

At operational levels of a business, tasks are physically constrained by space and time to a finite set of atomic actions over a finite and discrete set of resources. Even in a business context where jobs may enter production with irregular specifications, the list of known actions over a set of resources given the set of jobs to date, will be always be finite and discrete. This property ensures that the known list of atomic actions for each resource will be finite and enumerable. In many companies, section supervisors make use of this property to record machine settings and to formulate procedures for different jobs, towards creating consistent work specifications and high-quality products. At this operational level, detailed information is routinely gathered about atomic actions at each step of production, which could even be specified as a formal set of action operators.

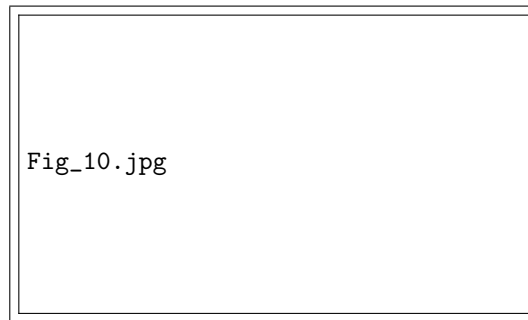


Figure 10: An automatic cable cutting machine described as a finite list of action operators, represents how a resource at the operational level of a business may be described with lists of action operators. The *pre_i* list represents a conjunction of predicate states that form a precondition; whereas, the *post_i* list represents a conjunction of predicate states in the post condition of action *Program_Job_i*.

Figure 10 gives a representation of how action operators may be formulated about one or more resources in a production line. In this particular example, a logical encapsulation of predicate states and action operators

have been described about a single non-human resource; however, such encapsulations may be defined about human resources and even whole subsections of a production line. This logical unit of production in such an encapsulation will be called a *production unit*. In addition, we will consider action operators in a production unit as *primitive* action operators, since they represent logically atomic actions in a business process.

For the remainder of this section we will consider as a case study a business environment at the operational level that has been logically divided into production units and those units have been represented in a database by business objects involving action operators and states as in Figure 10. We will refer to this database as *Production Unit* or *PU* database. Thereafter, we will explore what benefits may arise from this arrangement and in the context of the preceding theory on action k-PN templates.

5.1. The Production Plan Problem

A common problem for many businesses is the efficient adjustment of tools and resources for new lines of production, which we will refer to as the *production plan* problem. In specifying this problem, we will assume that the input (\mathbf{m}_{in}) and output (\mathbf{m}_{out}) of the new production line are prior knowledge and the problem is to derive a plan that transforms the input to the output of production. Ideally, the information systems of a business environment would automatically generate a solution to this problem; however, this problem may be difficult to solve if those systems are not optimised for the analysis. Since the PU database in this case study is optimised for planning, it can supply action operators to an automatic planning algorithm (e.g., STRIPS in Section 3.2) in solving the production plan problem.

Let's assume that the PU database in the case study has sufficient action operators so that *STRIPS* returned a list of models $M_\Sigma = [m_0, m_1, m_2, \dots, m_i, \dots, m_n]$ representing a plan for a new production line. Given this list, the plan to action k-PN mapping in Section 4.3 may be applied to generate a list of k-PN templates: $A_{MV} = [f_{kPN,0}, f_{kPN,1}, f_{kPN,2}, f_{kPN,3}, f_{kPN,4}, f_{kPN,5}]$, which also represents the new production line plan. A natural extension to the information systems of our case study would be a database of k-PN templates and k-PN template lists; we will call this database the *kPN* database. In the current example, the list A_{MV} and the combination $F_{kPN} = f_{kPN,0} \oplus f_{kPN,1} \oplus f_{kPN,2} \oplus f_{kPN,3} \oplus f_{kPN,4} \oplus f_{kPN,5}$ may be inserted into the kPN database. In fact, even logical sub-combinations may be inserted so that hierarchical compositions can also be queried. Given that the predicate $f_{kPN} \rightarrow (\mathbf{m}_{out} \mid \mathbf{m}_{in})$ can test the template f_{kPN} for any given initial state \mathbf{m}_{in} , a query for an existing plan that solves a new production line in the future may proceed as follows:

SELECT f_{kPN} FROM 'kPN database' WHERE $f_{kPN} \rightarrow (\mathbf{m}_{out} \mid \mathbf{m}_{in})$

In some cases, a production plan may only be partially represented in the PU or kPN database so that steps to solve the production plan problem will involve adding new data to the databases. For example, if the new production line is similar to a past production plan represented in the kPN database by the list of templates: $A_X = [f_{kPN,0}, f_{kPN,1}, f_{kPN,2}, f_{kPN,3}, \dots, f_{kPN,n}]$ then the list can be searched to determine at what point the plan will fail:

1. For each $f_{kPN,i} \in A_X$ indexed by i
2. If $f_{kPN,i} \rightarrow (\mathbf{m}_{out} \mid \mathbf{m}_{in}) = \text{FALSE}$ then return i

This algorithm can inform a business process of actions that can be reused and where production units need to be either added or modified. In the case where the kPN database is uninformative, planning algorithms can be devised to return partial plans from the PU database towards a similar analysis.

5.2. Workflow Model Creation

The kPN database can automatically inform the creation of workflow models. We recall from Section 4.2 that an action k-PN template is a template that reduces to a Petri Net (a k-PN) given an appropriate input value. In creating a Petri Net based workflow model (see Figure 11), the kPN database may supply action k-PN templates, where the reduction of a template to a k-PN that will consume \mathbf{m}_{in} and generate \mathbf{m}_{out} may be tested. That is, for each $f_{kPN,j}$, if $f_{kPN,j} \rightarrow (\mathbf{m}_{out} \mid \mathbf{m}_{in})$ then $f_{kPN,j}$ will reduce to a k-PN that will functions as the required task; otherwise, $f_{kPN,j}$ does not represent a sensible task in the workflow.

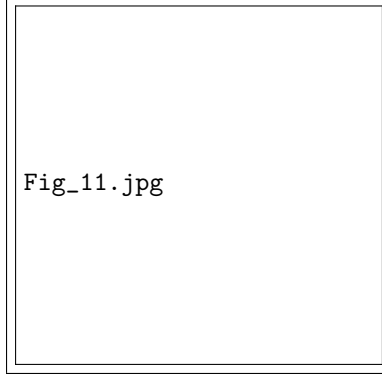


Figure 11: Given $f_{kPN,j} \rightarrow (\mathbf{m}_{out} \mid \mathbf{m}_{in})$, the action k-PN template $f_{kPN,j}$ reduces to a k-dimensional Petri Net that can function as a task in a Petri Net based workflow model.

If the kPN database contains template hierarchies derived through the associative property of \oplus , then workflow models can be decomposed to represent workflows at different levels of operation. Figure 12 involves the list of action k-PN templates $A_{MV} = [f_{kPN,0}, f_{kPN,1}, f_{kPN,2}, f_{kPN,3}, f_{kPN,4}, f_{kPN,5}]$ that was discussed in Section 4.4, where the associative hierarchy was derived from the expression: $F_{kPN} = (((f_{kPN,0} \oplus f_{kPN,1}) \oplus (f_{kPN,2} \oplus f_{kPN,3})) \oplus (f_{kPN,4} \oplus f_{kPN,5}))$. This associative hierarchy would typically represent a natural associative grouping of primitive actions into higher level tasks and task encapsulations aimed at informing different levels of management.

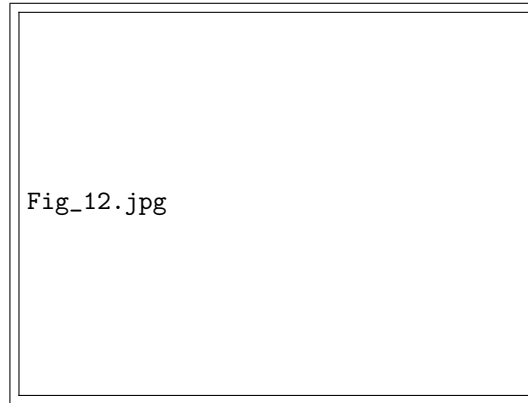


Figure 12: Given a suitable input value, a hierarchical association of action k-PN templates (top) reduces to four levels of k-PN chains (bottom) that can each represent a different level of operation in a workflow model.

Finally, given a set of action k-PN templates $\{f_{kPN,j}\}$ and variable vectors \mathbf{m}_x and \mathbf{m}_y over a set of multiplicity values, the set of predicates $\{f_{kPN,j} \rightarrow (\mathbf{m}_y \mid \mathbf{m}_x)\}$ can function as a set of efficient search heuristics that may even inform automatic workflow model generations from a kPN database. For example, the base rules for a depth-first search may involve expanding all \mathbf{m}_x where there exists an \mathbf{m}_y not yet visited so that $f_{kPN,j} \rightarrow (\mathbf{m}_y \mid \mathbf{m}_x)$.

6. Conclusion and Further Work

In this paper we have proven a deductive system that can be used to associate and transform information gathered from the operational levels of businesses for different levels of management. We envisage business

environments where section supervisors would maintain data about production units in information systems, in the form of action operators and action k-PN templates. Different levels of management could benefit from queries involving the combination operator (\oplus) and associative hierarchical compositions, programmed to return accurate data at appropriate levels of resolution. Such information systems may involve planning algorithms to solve or partially solve production plans, thus saving time and resources that would have been consumed in setup and dry production runs. We envisage from this work, workflow modelling systems that can inform the modeller of valid tasks given an existing business operations.

There are problems that have not been solved in this framework and remain as further work. For example, operators need to be investigated for combining action k-PN templates in parallel, which is a complex problem that will well exceed the scope of a single paper. A number of simplistic kinds of parallel combinations could be defined based only on the multiplicity values; however, none of these operators can account for issues of template duplications between operands that may occur in their hierarchies. Another issue relating to parallel template combinations concerns the relative time of actions represented by templates in the decompositions of operands and that time has not been represented in our current framework.

References

- [1] W. M. P. van der Aalst, K. M. van Hee, G. J. Houben, Modelling Workflow Management Systems with High-Level Petri Nets, in: G. de Michelis, C. Ellis, G. Memmi (Eds.), Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms, pp. 31–50.
- [2] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, A. P. Barros, Workflow Patterns, Distrib. Parallel Databases 14 (2003) 5–51.
- [3] N. Russell, A. H. ter Hofstede, W. M. van der Aalst, newYAWL: Specifying a Workflow Reference Language using Coloured Petri Nets, in: K. Jensen (Ed.), Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, pp. 107–126.
- [4] M. Jurič, B. Mathew, P. Sarang, Business process execution language for web services, Packt Publishing Ltd, 2006.
- [5] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-science: An overview of workflow system features and capabilities, Future Generation Computer Systems 25 (2009) 528–540.
- [6] A. ter Hofstede, W. van der Aalst, M. Adams, Modern Business Process Automation: YAWL and its Support Environment, Springer-Verlag, 2009.
- [7] W. van der Aalst, A. H. M. T. Hofstede, M. Weske, Business Process Management: A Survey, in: Proceedings of the 1st International Conference on Business Process Management, LNCS, volume 2678, Springer-Verlag, 2003, pp. 1–12.
- [8] B. Dougherty, J. White, D. C. Schmidt, Model-driven auto-scaling of green cloud computing infrastructure, Future Generation Computer Systems (2011).
- [9] Y. Qu, A predicate-ordered logic for knowledge representation on the web, Future Generation Computer Systems 20 (2004) 19–26.
- [10] H. Guo, R. Brown, R. Rasmussen, Human resource behaviour simulation in business processes, Information Systems Development (2011).
- [11] W. Van Der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer-Verlag New York Inc, 2011.
- [12] W. M. P. van der Aalst, On the automatic generation of workflow processes based on product structures, Computers in Industry 39 (1999) 97–111.
- [13] W. M. P. V. der Aalst, Timed Coloured Petri Nets and their Application to Logistics, Ph.D. thesis, Eindhoven University of Technology, 1992.
- [14] G. Rozenburg, J. Engelfriet, Elementary Net Systems, in: W. Reisig, G. Rozenberg (Eds.), Lectures on Petri Nets I: Basic Models - Advances in Petri Nets, volume 1491 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 12–121.
- [15] N. R. Adam, V. Atluri, W. kuang Huang, Modeling and Analysis of Workflows using Petri Nets, Journal of Intelligent Information Systems 10 (1998) 131–158.
- [16] W. van der Aalst, L. Aldred, M. Dumas, T. A. H. M. Hofstede, Design and Implementation of the YAWL System, Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04) (2004).
- [17] K. Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, volume Volume 1 of *EATCS Series*, Springer Verlag, 2003.
- [18] K. Jensen, An Introduction to the Theoretical Aspects of Coloured Petri Nets, in: A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Springer-Verlag, London UK, 1994, pp. 230–272.
- [19] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, second edition, 2002.
- [20] R. E. Fikes, N. J. Nilsson, Strips: A New Approach to the Application of Theorem Proving to Problem Solving, Artificial Intelligence 2 (1971) 189–208.